Authors: Maximelio Finch, Sabrina Hubbard, Diego Rivera

Professor Chu

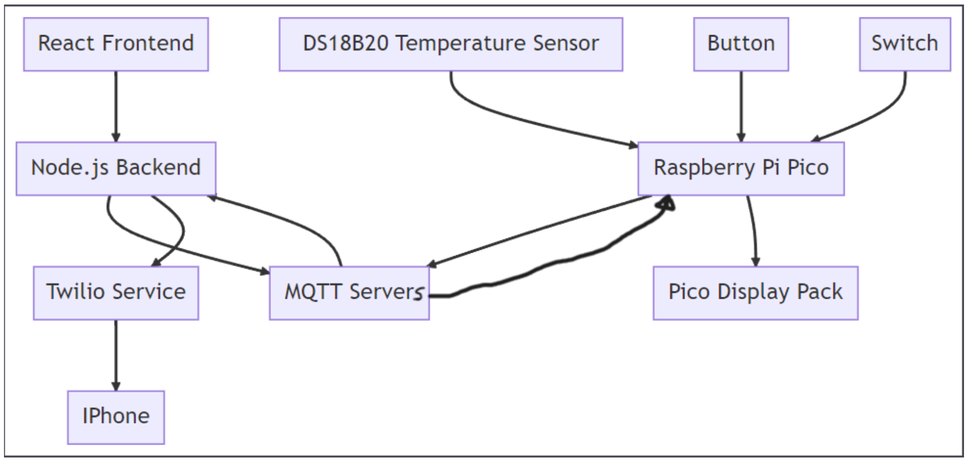
ECE:4880 Prin of Elec and Comp Sen Design

Project Name: Web-Connected Thermometer

Introduction:

In the Web-Connected Thermometer project, our objective was to create a sophisticated temperature monitoring system that offers both real-time data visualization and user-friendly interaction. At the heart of our device is the Raspberry Pi Pico (Pico W), complemented by the high-resolution Pimoroni Display (PIM543) and the precision-driven DS18S20 1-Wire Digital Thermometer. To enhance user experience and provide remote access capabilities, we developed a web interface using React, with the coding environment facilitated by Visual Studio Code. The software foundation for this project was built upon CircuitPython for the hardware components and JavaScript for the web interface.

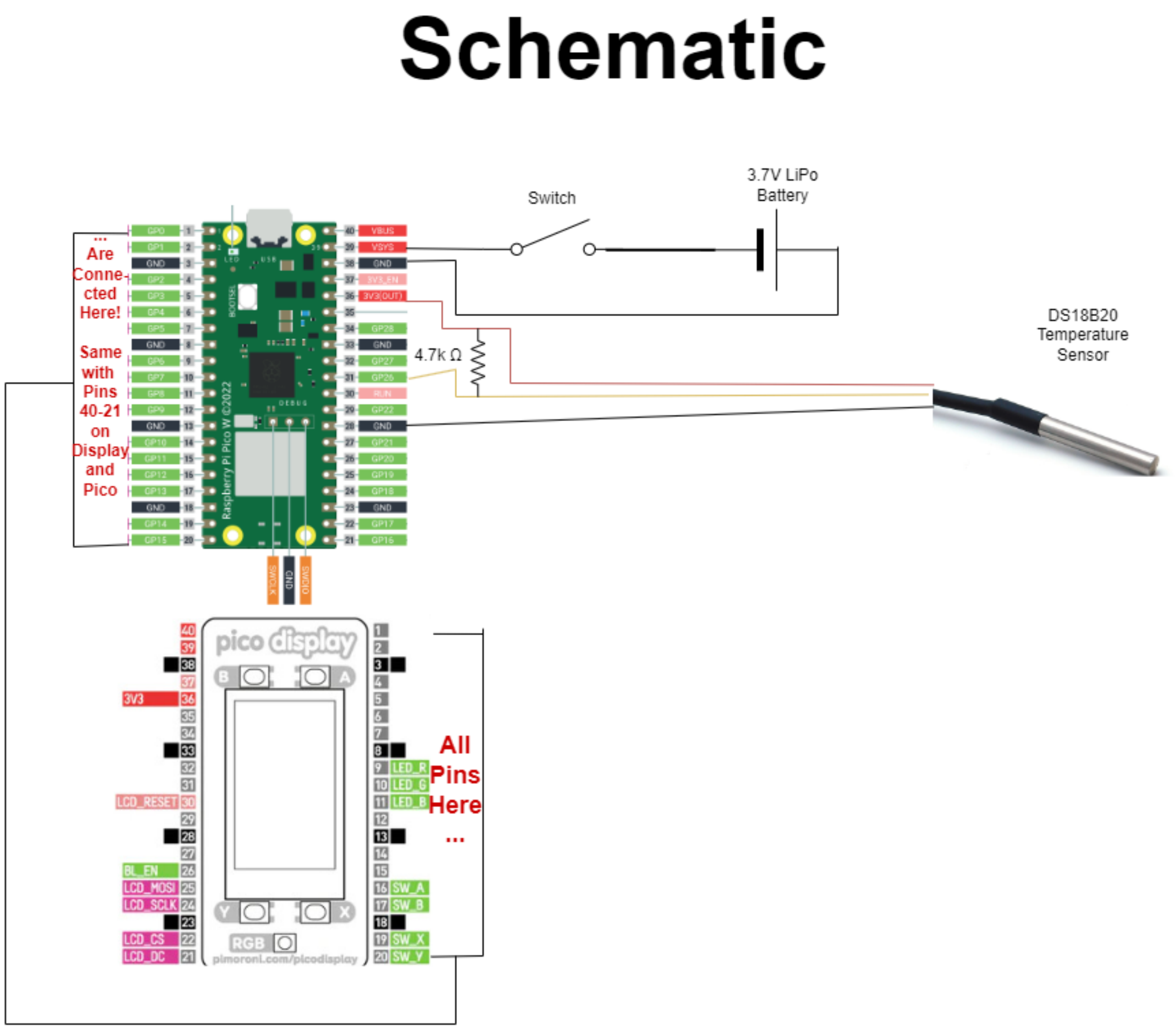
High-Level Block Diagram:



1. **React Frontend:** User interface for viewing real-time temperature data, changing temperature messages, altering phone numbers, and viewing live graphing of temperature data.
2. **Node.js Backend:** Connects the React frontend to the MQTT server, facilitates temperature data flow, and interacts with the Twilio service.
3. **MQTT Servers:** Manages communication between the Raspberry Pi Pico and the Node.js backend.
4. **Raspberry Pi Pico:** Microcontroller that reads data from the DS18B20 temperature sensor, displays it on the Pico Display Pack, and communicates with the MQTT server.
5. **DS18B20 Temperature Sensor**: Measures temperature and sends data to the Raspberry Pi Pico.
6. **Button:** Triggers the Pico Display Pack to show the temperature when pressed.
7. **Switch:** Acts as an on/off switch for the thermometer system.
8. **Pico Display Pack:** Displays the temperature when the button is pressed.
9. **Twilio Service:** Sends text messages to a specified phone number based on temperature thresholds.

10. **Phone:** Receives text messages from the Twilio service when temperature thresholds are breached.

Schematic:

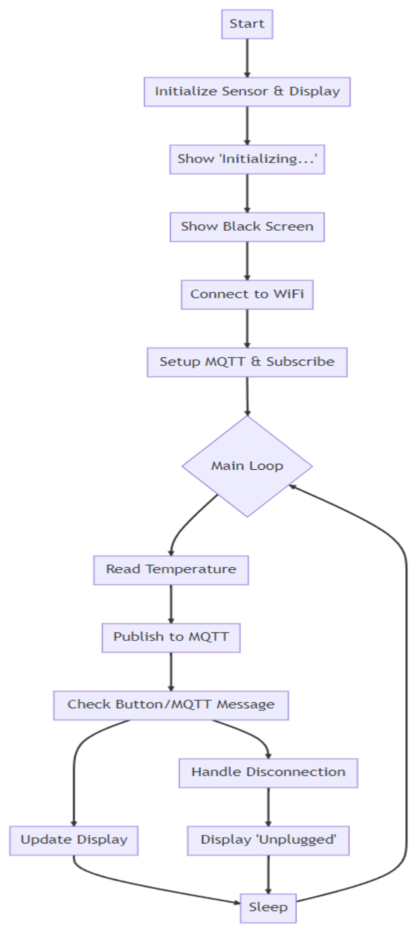


Hardware Schematic Description:

1. **Raspberry Pi Pico (Pico W):** The central microcontroller board serves as the heart of the system, interfacing with all the hardware components.
2. **Power Supply:**
    a. **3.7V LiPo Battery:** Provides the primary power source for the system.
    b. **Switch:** Positioned in series between the LiPo battery and the Pico's VSYS pin. This switch allows for manual power control, letting the user turn the Pico on and off.
3. **DS18S20 High-Precision 1-Wire Digital Thermometer:**
    a. **Data Connection:** Linked to the GP26 pin on the Pico, enabling the Pico to read temperature data.
    b. **Power:** Draws power from the 3.3V pin on the Pico.

4. **Pimoroni Pico Display (PIM543):**
    a. Interfaces directly with the Pico, serving as the visual output for the system. It displays temperature readings and potential system messages or alerts.
    b. Button A: Integrated into the display pack. When pressed, it toggles the display on and off, allowing the user to manually control the visibility of the temperature readings and system messages.
5. **Wi-Fi Module (embedded in the Pico W):** Enables wireless communication, allowing the Pico to connect to external networks and services.

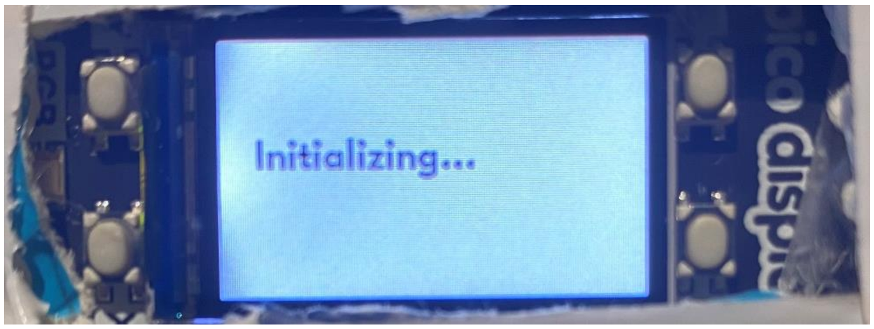
Firmware Block Diagram:



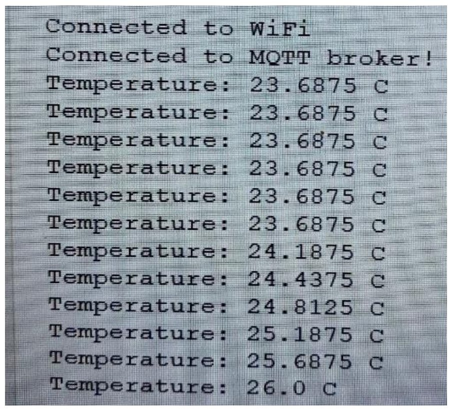
1. **Initialization:** The system initializes the temperature sensor and the display. The drivers for the Pico display are **custom.**
2. **Display Initialization Message:** A brief "Initializing..." message is shown to the user.
3. **Black Screen:** After the initialization message, the display turns black.
4. **Wi-Fi Connection:** The system connects to the Wi-Fi network.
5. **MQTT Setup:** MQTT is set up, and the system subscribes to the topic "/display".
6. Main Loop:
    a. **Read Temperature:** The system continuously reads the temperature from the DS18B20 sensor.
    b. **Publish to MQTT:** The temperature readings are published to an MQTT topic.
    c. **Check Button/MQTT Message:** The system checks if the display button is pressed or if there's a message from MQTT to turn the display on/off.
    d. **Update Display:** If the button is pressed or an "ON" message is received via MQTT, the temperature is displayed. Otherwise, the screen will remain black.
    e. **Handle Disconnection:** If the sensor gets unplugged, the system handles the disconnection error.
    f. **Display 'Unplugged':** An "Unplugged" message is shown on the display to inform the user about the sensor disconnection.
7. **Sleep:** The system goes into a brief sleep mode before repeating the main loop.

Overall Operation and Supporting Images:

(1) The system starts by initializing the temperature sensor and the display. Upon initialization, the display briefly shows an "Initializing..." message before transitioning to a black screen.

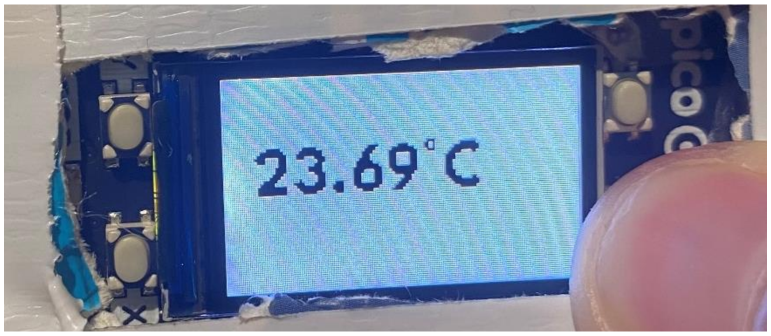


(2) The system then connects to the Wi-Fi network and sets up MQTT (Message Queuing Telemetry Transport) for communication. Once connected, it subscribes to the 'Pico/display' topic to receive messages. In the main loop, the system continuously monitors the temperature using the DS18B20 sensor and publishes this data to the MQTT broker.

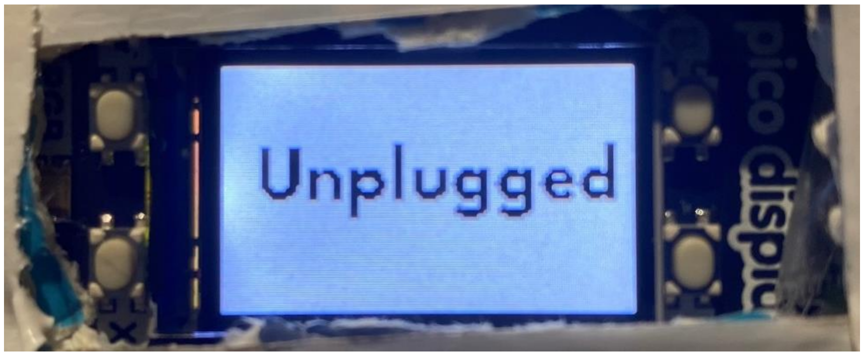


```
Connected to WiFi
Connected to MQTT broker!
Temperature: 23.6875 C
Temperature: 23.6875 C
Temperature: 23.6875 C
Temperature: 23.6875 C
Temperature: 23.6875 C
Temperature: 23.6875 C
Temperature: 24.1875 C
Temperature: 24.4375 C
Temperature: 24.8125 C
Temperature: 25.1875 C
Temperature: 25.6875 C
Temperature: 26.0 C
```

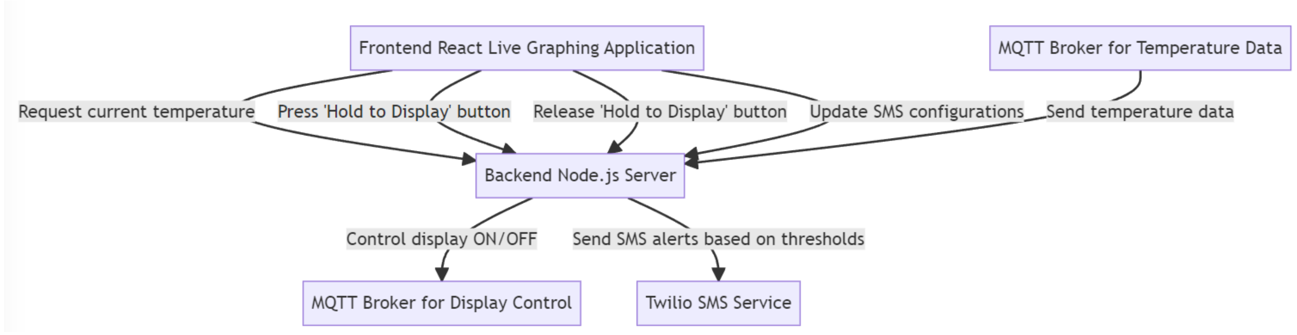
(3) The display can be toggled on or off using the Button A on the Pico display or through an MQTT message.



(4) If the temperature sensor is disconnected, the system handles this error by updating the display with an "Unplugged" message. The system then sleeps for approximately 0.06 seconds before repeating the loop.



Webapp Block Diagram:


Frontend React Live Graphing Application
MQTT Broker for Temperature Data
Request current temperature
Press 'Hold to Display' button
Release 'Hold to Display' button
Update SMS configurations
Send temperature data
Backend Node.js Server
Control display ON/OFF
Send SMS alerts based on thresholds
MQTT Broker for Display Control
Twilio SMS Service

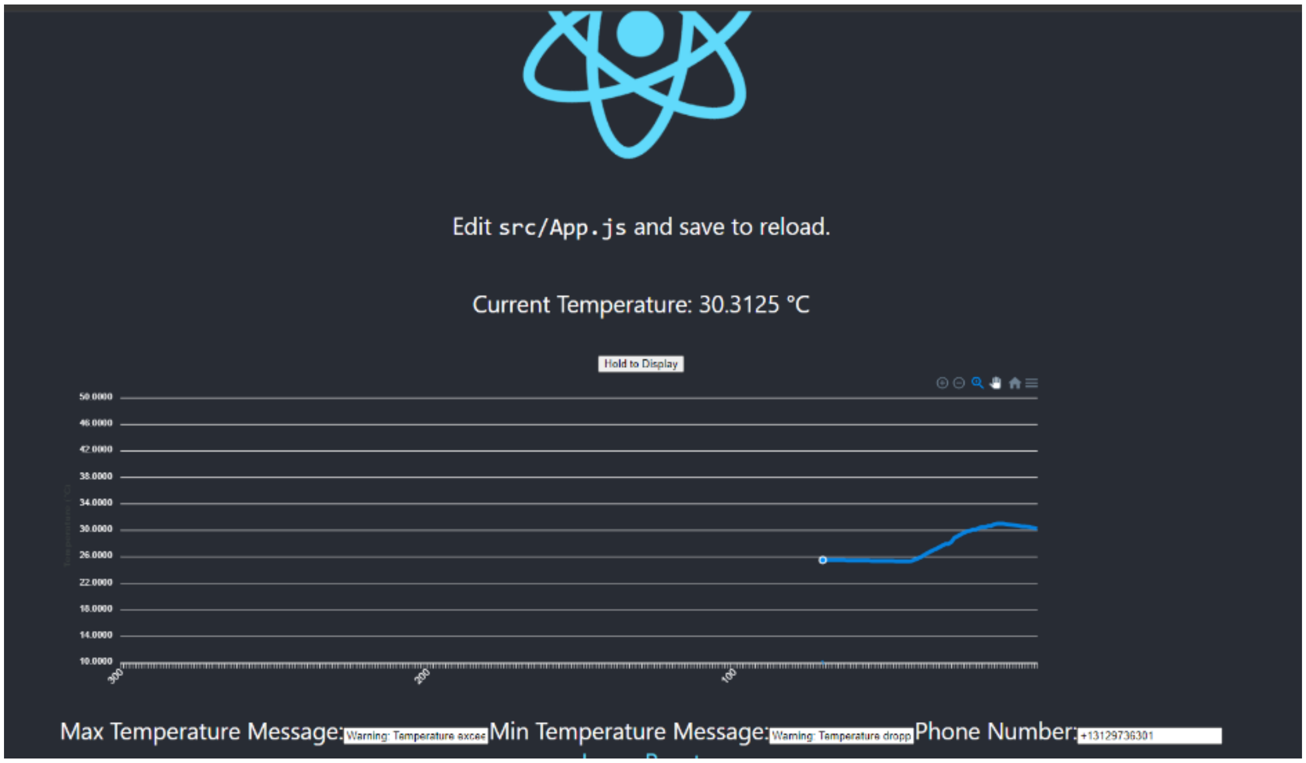
1. **Frontend React Application:** This is the user interface where users can:
    a. **View the current temperature:** The temperature is displayed in real-time, updating every second.
    b. **Graphical Representation:** The React application uses the ApexCharts library to graphically represent the temperature data. The graph displays the most recent 300 temperature data points, updated in real-time. The x-axis represents the data points, while the y-axis represents the temperature in either Celsius (°C) or Fahrenheit (°F). The graph adjusts its size responsively based on the screen size.
    c. **Press the 'Hold to Display' button:** When this button is pressed and held, the temperature is displayed on an external device.
    d. **Release the 'Hold to Display' button:** Releasing the button stops the display on the external device.
    e. **Update SMS configurations:** Users can set the maximum and minimum temperature thresholds and specify the phone number to which alerts should be sent if the temperature goes beyond these thresholds.
2. **Backend Node.js Server:** This server handles various tasks:
    a. **Receives Temperature Data:** The server receives temperature data from the MQTT broker.
    b. **Controls the Display:** It controls the display (ON/OFF) by sending commands to another MQTT broker based on user interactions from the front-end.
    c. **Sends SMS Alerts:** If the temperature exceeds or drops below the set thresholds, the server sends SMS alerts through the Twilio service.
3. **MQTT Broker for Temperature Data:** This broker sends temperature data to the Node.js server.

4. **MQTT Broker for Display Control:** The Node.js server sends commands to this broker to control the display (turn it ON or OFF).
5. **Twilio SMS Service:** The Node.js server uses this service to send SMS alerts when the temperature exceeds or drops below the set thresholds.

Overall Operation and Supporting Images:

1. The React frontend frequently requests the current temperature from the Node.js backend.
2. The backend, in turn, receives temperature data from the MQTT broker.
3. When a user interacts with the "Hold to Display" button on the front-end, the back-end sends commands to another MQTT broker to control a display.
4. The front end also allows for the temperature messages to be changed.

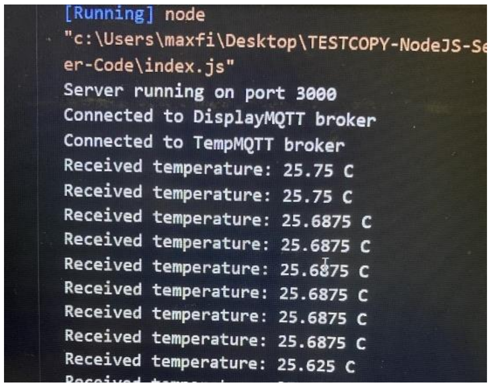
React Front-End User Interface: Screenshot of the UI seen by the user on a computer. The "Hold to Display" button and Temperature Text Message editable textboxes are visible.


Edit `src/App.js` and save to reload.

Current Temperature: 30.3125 °C

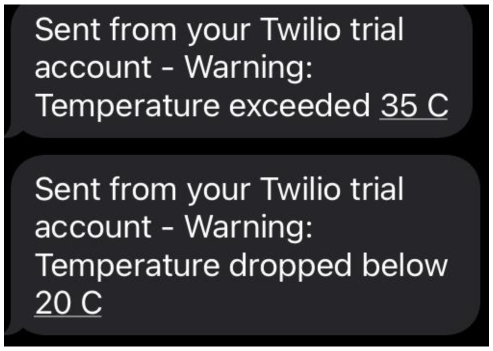
Max Temperature Message: Warning: Temperature excee Min Temperature Message: Warning: Temperature dropp Phone Number: +13129736301

Node.js Back-end: Screenshot of the data the server receives from the Pico via MQTT.



```
[Running] node
"c:\Users\maxfi\Desktop\TESTCOPY-NodeJS-Se
er-Code\index.js"
Server running on port 3000
Connected to DisplayMQTT broker
Connected to TempMQTT broker
Received temperature: 25.75 C
Received temperature: 25.75 C
Received temperature: 25.6875 C
Received temperature: 25.6875 C
Received temperature: 25.6875 C
Received temperature: 25.6875 C
Received temperature: 25.6875 C
Received temperature: 25.6875 C
Received temperature: 25.625 C
```

<u>Twilio Messages:</u> If the temperature exceeds or falls below certain thresholds, the backend sends an SMS alert using the Twilio service.

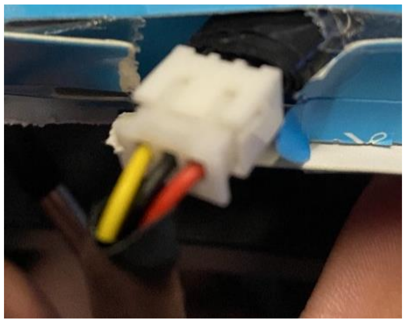


Sent from your Twilio trial account - Warning: Temperature exceeded 35 C

Sent from your Twilio trial account - Warning: Temperature dropped below 20 C

<u>Mechanical Design:</u>

Switch: The mechanical switch mentioned in the schematic is used for turning the pico-thermometer on/off.



Temperature Sensor Connector: Unpluggable connector for the DS18B20 temperature sensor.

Overall Physical Enclosure: Visual for how the schematic was implemented in the physical construction.

<u>Design Process and Experimentation:</u>

1. **Switch from MicroPython to CircuitPython:**
    a. **Alternative:** Continue using MicroPython.
    b. **Trade-offs:** CircuitPython had supported libraries for our DS18S20 1-Wire Digital Thermometer, which would reduce development time. MicroPython, while versatile, lacked the necessary libraries.
    c. **Decision:** Switch to CircuitPython for better sensor support.
2. **Creation of Pimoroni Pico Display Drivers:**
    a. **Alternative:** Use generic drivers or switch to another display.
    b. **Trade-offs:** Designing our own drivers was time-consuming but allowed for tailored customizations. Using generic drivers or another display might not have met our specific project needs.
    c. **Decision:** Design custom drivers for the Pimoroni Pico Display.
3. **Wiring Issues with the 3.7V 500mAh LiPo Battery:**
    a. **Experiment:** Troubleshoot the wiring and connections.
    b. **Observations:** Identified issues in the initial wiring setup.
    c. **Decision:** Rectified the wiring to ensure a stable power source.
4. **Design Evolution:**
    a. Alternative Designs:
        i. **Initial Design:** All components on a breadboard.
        ii. **Intermediate Design:** Sensor, display, and switch connected directly to the Pico.
        iii. **Trade-offs:** Moving from a breadboard to soldering components directly to the Pico reduced complexity and size but increased the permanence of the setup.
        iv. **Decision:** Final design housed in a cardboard box for aesthetics and user-friendliness.
5. **MQTT Connectivity Issues:**
    a. **Experiment:** Dive deeper into MQTT documentation and setup.
    b. **Observations:** Initial unfamiliarity led to setup issues. We unnecessarily sent the IP of the host computer to the Pico to connect them.
    c. **Decision:** After understanding MQTT better, established a stable connection.
6. **Switch from IFTTT to Twilio for Temperature Alerts:**

a. **Alternative:** Continue using IFTTT for temperature alerts.
b. **Trade-offs:** IFTTT offers a wide range of applets and connections but might not provide the direct SMS capabilities or reliability that Twilio offers. Twilio, while potentially more costly, offers robust SMS services.
c. **Decision:** Switch to Twilio for reliable temperature alert messaging.

Test Report:

Test Description	**Objective**	**Outcome**
Sensor Robustness Test	Ensure sensor durability when dropped	Passed
Button Response Test	Verify button response time	Passed
Internet Connectivity Test	Confirm internet access	Passed
Temperature Accuracy Test	Validate temperature accuracy	Passed
Text Message Alert Test	Verify temperature alerts	Passed
Web Temperature Toggle Test	Confirm temperature conversion from F ° to C °	Failed
Web Graph Update Test	Ensure accurate update time of 1s	Passed
Third Box Durability Test	Confirm ability to with stand small drops	Passed
Connector security	Ensure strong female to male connection when dropped	Passed
Pico-to-MQTT	Confirm connection with Pico and MQTT Broker	Passed
Node.js-to-MQTT	Confirm connection with Node.js and MQTT Broker	Passed
Virtual Button Press and Delay Test	Confirm React button press triggers the Pico display to turn on as described in the requirements.	Passed
Pimoroni Display Pack Driver Test	Verify custom drivers work as a wrapper for the ST7789 display driver.	Passed
Unplugged Sensor Test	Ensure the sensor being unplugged does not terminate any parts of the software and can be plugged back in to resume normal operation.	Passed
Change of Phone Number	Check whether changing the phone number on the React interface changes the number the messages are sent to.	Passed

Test Description	Objective	Outcome
Change of High Temperature Text	Checks whether changing the High Temperature Text on the React interface changes the message sent to the iPhone.	Passed
Change of Low Temperature Text	Checks whether changing the Low Temperature Text on the React interface changes the message sent to the iPhone.	Passed
API Call Frequency	Confirm the code blocks in the Node.js code for rapid API calls work as intended.	Passed
Node.js-React Connectivity	Ensure the connection between the Node.js backend and react front-end running on two different ports on the host computer.	Passed
Apex Chart Optimization Test	Removed animation functionality from the live charting library to smoother transitions.	Passed
Twilio Number Registration	Register Twilio Number as a Toll-Free number for no interruptions in operation.	Passed
Switch and LiPo Battery Test	Test if the switch can properly toggle the LiPo battery on and off, and the battery successfully powers the Pico to run its code.	Passed
Proper Delay Test	Test if .06125 milliseconds deliver the required response time in the Pico firmware's main loop.	Passed
React Error Handling Test	Ensure that different errors caused by physical interruptions of the Pico (turning the system off, unplugging the sensor) display the proper error types to the user on the front end.	Passed
React Button Handler Test.	React button handlers for button press, SMS message change, and Phone Number change properly triggers the respective HTTP request to the Node.js server.	Passed

Project Retrospective:

1. **Project Management:** We integrated the Agile project management process into the development of the Web-Connected Thermometer lab. This approach facilitated flexibility and responsiveness to challenges. Key Agile practices, such as the creation of a project backlog, sprint planning, weekly standups, continuous integration, iterative development, feedback incorporation, and incremental testing were pivotal in our adaptive methodology. The inception

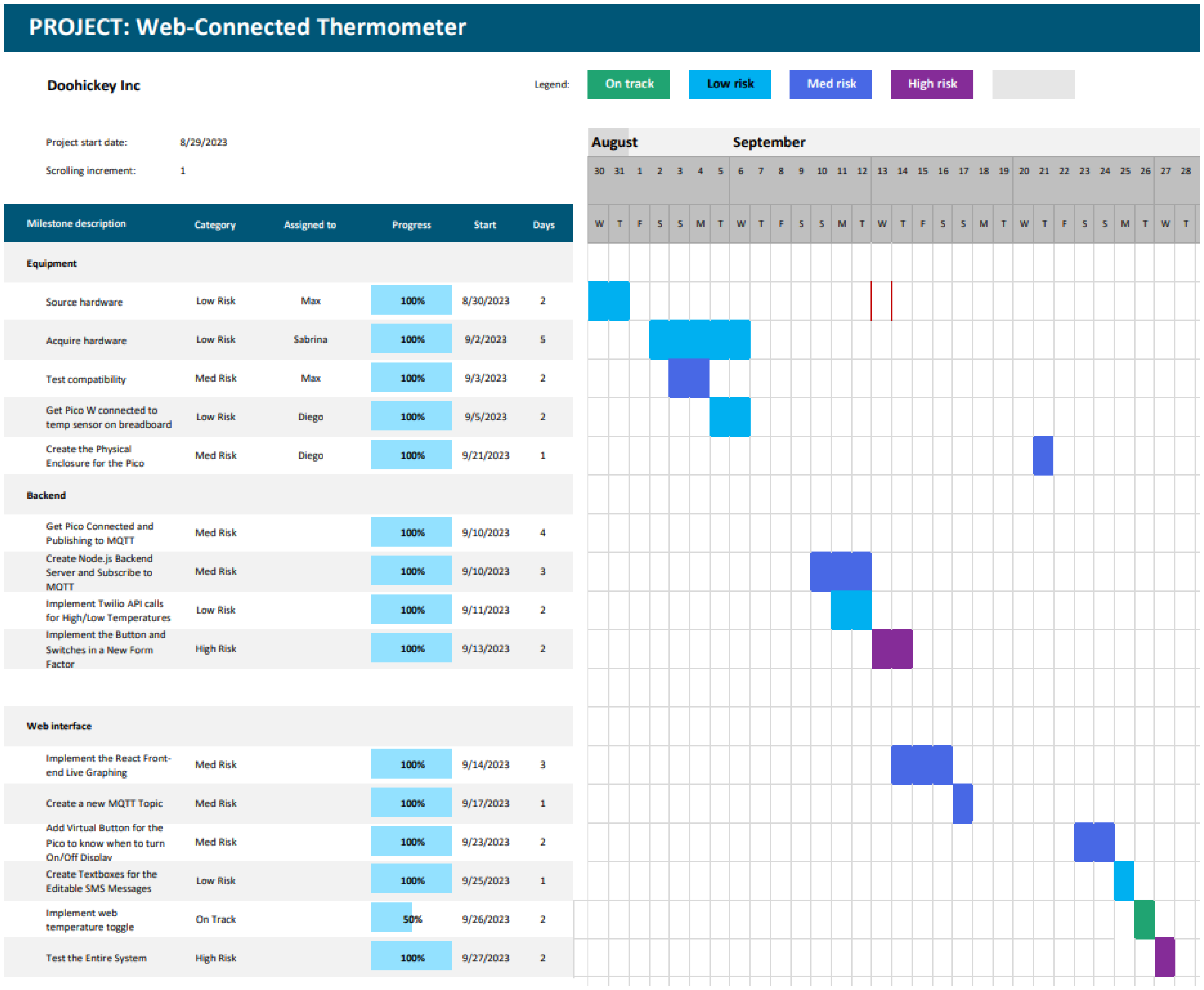
of the project saw the formation of a comprehensive backlog, cataloging tasks and features spanning hardware design, software development, and testing. We utilized sprint planning to segment the project into manageable iterations. Our weekly standups, adapted from daily sessions, optimized our discussion of progress and obstacles, aligning with our academic and professional commitments. Continuous integration ensured consistent amalgamation of code changes, promoting seamless interfacing. Each sprint marked an era of iterative development, characterized by the incremental introduction of features and functionalities, enriched by feedback from team members and recalibration of priorities in real-time. Documentation was meticulously maintained, and incremental testing underscored our focus on validating components iteratively rather than an end-to-end validation, enhancing adaptability and communication. The Agile approach not only streamlined our Web-Connected Thermometer project but also elevated its resilience to complexities and uncertainties, culminating in a refined, end-user centric product.

2. **Summary of Project Outcome and Critical Assessment:** The switch from MicroPython to CircuitPython was pivotal, enhancing sensor support due to the availability of libraries for our DS18S20 1-Wire Digital Thermometer. While MicroPython was versatile, the explicit support from CircuitPython streamlined our development process. The decision to create custom drivers for the Pimoroni Pico Display, despite being time-intensive, proved beneficial, offering tailored customization that generic drivers or an alternative display could not provide. This bespoke approach ensured the display met our specific project requirements and expectations. We encountered wiring issues with the 3.7V 500mAh LiPo Battery, which were resolved through meticulous troubleshooting, ensuring a stable and reliable power source for the project's components. Each wiring correction augmented the project's reliability and performance. Our design evolved from an initial breadboard setup to a final, user-friendly design housed in a cardboard box. This progression, although making the setup more permanent, minimized complexity and optimized aesthetics and functionality, enhancing user interaction. MQTT connectivity presented a challenge due to our initial unfamiliarity with its documentation and setup. A deeper engagement and understanding facilitated a stable and efficient connection, overcoming initial hurdles and ensuring seamless data transmission. Lastly, the transition from IFTTT to Twilio for temperature alerts was marked by Twilio's superior reliability in SMS services, albeit potentially higher costs. While IFTTT offered diverse applets, Twilio's robust SMS capabilities assured direct and reliable temperature alert messaging. In retrospect, each decision and alteration made during the design process and experimentation was instrumental in navigating challenges and optimizing the project's outcome. The tailored approaches, though occasionally time-consuming, ensured that the final product was not only functional but also user-centric, reliable, and efficient in data handling and alert messaging. The critical assessment underscores a successful amalgamation of informed decisions, adaptive strategies, and technical refinements that collectively enhanced the project's efficacy and user experience.
3. **Incomplete Goals and Mitigation:** The project's completion was marked by the omission of a web interface toggle allowing the temperature sensor reading to switch between Fahrenheit and Celsius. This limitation stemmed from a cautious approach to code modification and time constraints, as we opted for preservation over innovation in the final sprint. Our reluctance to alter the code to accommodate this feature was a bid to maintain the existing functionality intact. We recognized that the trade-off between complete feature integration and meeting

overarching project goals is intrinsic to project management. Balancing the triple constraints of time, cost, and scope is fundamental; altering one invariably impacts the others. In hindsight, to preclude such omissions, our future sprints will be informed by enhanced preparatory measures. We'll enhance our anticipation and responsiveness to potential issues by fortifying our time and functionality forecasts, ensuring that subsequent projects are both comprehensive and timely.

4. **Workload distribution:** This can be improved in further labs. Team members played to their strengths: EE aided in hardware selection, CS in front-end development, and CS/EE in embedded system implementation. All team members aided in the report and documentation.
5. **Deviations From Projections**: In the execution of the web interconnected thermometer project, several deviations from the initial projections were encountered. The software development phase, initially slated for 2 weeks, extended to 3 weeks due to unexpected bugs and complexities. Resource availability became a challenge when key team members were pulled into other urgent projects, resulting in a slowdown in progress. The integration of technological components, projected to be seamless, faced complications due to unanticipated incompatibilities, necessitating additional resolution time. Vendor delivery delays also posed a challenge; materials expected in the first couple days were pushed to the second week due to unforeseen holdups. Each deviation required strategic adjustments to maintain project integrity and progress, underscoring the necessity for adaptability and proactive planning in project management.

Gantt Chart:



PROJECT: Web-Connected Thermometer

Doohickey Inc

Legend: On track | Low risk | Med risk | High risk

Project start date: 8/29/2023

Scrolling increment: 1

August September

Milestone description	Category	Assigned to	Progress	Start	Days
Equipment					
Source hardware	Low Risk	Max	100%	8/30/2023	2
Acquire hardware	Low Risk	Sabrina	100%	9/2/2023	5
Test compatibility	Med Risk	Max	100%	9/3/2023	2
Get Pico W connected to temp sensor on breadboard	Low Risk	Diego	100%	9/5/2023	2
Create the Physical Enclosure for the Pico	Med Risk	Diego	100%	9/21/2023	1
Backend					
Get Pico Connected and Publishing to MQTT	Med Risk		100%	9/10/2023	4
Create Node.js Backend Server and Subscribe to MQTT	Med Risk		100%	9/10/2023	3
Implement Twilio API calls for High/Low Temperatures	Low Risk		100%	9/11/2023	2
Implement the Button and Switches in a New Form Factor	High Risk		100%	9/13/2023	2
Web interface					
Implement the React Front-end Live Graphing	Med Risk		100%	9/14/2023	3
Create a new MQTT Topic	Med Risk		100%	9/17/2023	1
Add Virtual Button for the Pico to know when to turn On/Off Display	Med Risk		100%	9/23/2023	2
Create Textboxes for the Editable SMS Messages	Low Risk		100%	9/25/2023	1
Implement web temperature toggle	On Track		50%	9/26/2023	2
Test the Entire System	High Risk		100%	9/27/2023	2

Appendix & References:

1. Raspberry Pi Pico: Official Documentation
2. DS18B20 Temperature Sensor: Technical Details
3. CircuitPython Libraries: Official Documentation
4. Twilio: Official Documentation

Software Libraries:

1. **Pico Code (Python):**
   a. **Wi-Fi:** Used to manage Wi-Fi connections on the Raspberry Pi Pico.
   b. **socketpool:** Provides a pool of sockets for network communication.
   c. **adafruit_minimqtt:** A minimal MQTT library for CircuitPython.
   d. **board:** Provides access to basic board-level hardware.
   e. **displayio:** Used for managing displays on CircuitPython devices.

    f. time: Provides time-related functions.
    g. **adafruit_onewire.bus:** Provides functionality for the OneWire bus, which is a communication protocol. Particularly for the DS18B20 Temperature Sensor.
    h. **adafruit_ds18x20:** Library for the DS18x20 family of temperature sensors.
    i. **adafruit_bitmap_font:** Used for loading bitmap fonts.
    j. **adafruit_display_text:** Provides text display capabilities on CircuitPython devices.
    k. **pimoroni_pico_display:** Library that **<u>we created</u>** for managing the Pimoroni Pico Display.

2. **Node.js Backend Code (JavaScript):**
    a. **express:** A web application framework for Node.js, designed for building web applications and APIs.
    b. **body-parser:** Middleware for parsing incoming request bodies in a middleware before handlers.
    c. **Twilio:** Client library for using the Twilio API, which provides communication capabilities like SMS.
    d. **MQTT:** MQTT client library for Node.js.
    e. **Cors:** Middleware that can be used to enable Cross-Origin Resource Sharing (CORS) in Express applications.
3. **React Front-end Code (JavaScript):**
    a. **React:** A JavaScript library for building user interfaces.
    b. **Axios:** Promise-based HTTP client for making HTTP requests from Node.js or XMLHttpRequests from the browser.
    c. **ApexCharts:** A modern charting library that helps developers to create interactive and beautiful charts.